BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

IN THE SPECIFICATION

Please replace paragraph [0009] with the following:

[0009] The floating point unit 135 is used for storing floating point data and includes a set of floating point registers (also termed as the floating point register file) 145, a set of tags 150, and a floating point status register 155. The set of floating point registers 145 includes eight registers labeled ~~R0~~ RØ to R7 (the notation Rn is used herein to refer to the physical location of the floating point registers). Each of these eight registers is 80 bits wide and contains a sign field (bit 79), an exponent field (bits [78:64]), and a mantissa field (bits [63:0]). The floating point unit 135 operates the set of floating point registers 145 as a stack. In other words, the floating point unit 135 includes a stack referenced register file. When a set of register is operated as a stack, operations are performed with reference to the top of the stack, rather than the physical locations of the registers in the set of floating point registers 145 (the notation STn is used herein to refer to the relative location of the logical floating point register n to the top of the stack). The floating point status register 155 includes a top of stack field 160 that identifies which register in the set of floating point registers 145 is currently at the top of the floating point stack. In FIG. 1, the top of stack indication identifies a register 165 at physical location R4 as the top of the stack.

Please replace paragraph [0035] with the following:

[0035] **FIG. 2** is a flow diagram illustrating the execution of an instruction by the Pentium ~~processor;~~processor;

Please replace paragraph [0044] with the following:

[0044] ~~**FIG. 7A** is a block diagram illustrating an expanded view of a portion of the floating point stack reference file from FIG. 6A according to embodiments of the invention;~~ FIG. 7A is a flow diagram illustrating a portion of a method, in accordance with one embodiment of the invention, for executing packed data instructions on a set of registers that are aliased on a set of floating point registers in a manner that is compatible with existing software, that is invisible to various operating system techniques, that promotes good programming practices, and that may be practiced using the hardware arrangement of FIG. 6A;

Please replace paragraph [0045] with the following:

[0045] ~~**FIG. 7B** is a block diagram illustrating an expanded view of a portion of the floating point stack reference file from FIG. 6A according to embodiments of the invention;~~ FIG. 7B is a flow diagram illustrating another portion of the method partially illustrated in FIG. 7A;

Please replace paragraph [0046] with the following:

[0046] ~~**FIG. 7C** is a block diagram illustrating an expanded view of a portion of the floating point stack reference file from FIG. 6A according to embodiments of the invention;~~ FIG. 7C is a flow diagram illustrating the remainder of the method partially illustrated in FIGS. 7A and 7B;

Please replace paragraph [0075] with the following:

[0075] In prior art floating point programming practice using the Intel architecture processor, it is common to terminate blocks of floating point code by an operation or operations which clear the floating point state. Irrespective of whether partial and/or minimal context switching is used, the floating point state is left in a clear condition upon the termination of a first block of floating point code. Therefore, the EMMS instruction is intended to be used in packed data sequences in order to clear the packed data state. The ~~EMNS~~ EMMS instruction should be executed after a block of packed data code. Thus, a processor implementing the methods and apparatus described here retains full compatibility with prior art floating point processors using the Intel Architecture processor, but yet, also have the capability of executing packed data instructions which, if programmed with good programming techniques and appropriate housekeeping (clearing the state before transitions between packed data code and floating point code), allow transitions between packed data and floating point code without adversely affecting either the floating point or packed data state.

Please replace paragraph [0121] with the following:

[0121] The decode/execution unit 575 is shown containing an instruction set 580 that includes packed data instructions. While these packed data instructions can be implemented to perform any number of different operations. For example, these packed data instructions, when executed, could cause the processor to perform packed floating point operations and/or packed integer operations. In one embodiment these packed data instructions are those described in "A Set of Instructions for Operating on Packed Data," filed on Aug. 31, 1995, U.S. Ser. No. 08/521,360, abandoned. In addition to the packed

data instructions, the instruction set 580 can include new instructions and/or instructions similar to or the same as those found in existing general purpose processors. For example, in one embodiment the processor 505 supports an instruction set which is compatible with the Intel processor architecture instruction set used by existing processors, such as the Pentium processor.

Please replace paragraph [0126] with the following:

[0126] FIG. 6A is a block diagram illustrating an apparatus for aliasing the packed data register state on the floating point state using two separate physical register ~~file~~ files according to one embodiment of the invention. Since these two physical register files are aliased, they logically appear to software executing on the processor as a single logical register file. FIG. 6A shows a transition unit 600, a floating point unit 605, and packed data unit 610. Floating point unit 605 is similar to floating point unit 135 of FIG. 1. Floating point unit 605 includes a set of floating point registers 615, a set of tags 620, a floating point status register 625 and a floating point stack reference unit 630. In one embodiment, the floating point unit 605 includes eight registers (labeled RØ to R7). Each of these eight registers is 80 bits wide and contains a sign field, an exponent field and an mantissa field. The floating point stack reference unit 630 operates the set of floating point registers 615 as a stack. The floating point status register 625 includes a top of stack field 635 for storing the top of stack indication. As previously described, the top of stack indication identifies which register in the set of floating point registers 615 is currently the top of the floating point stack. In FIG. 6A, the top of stack indication identifies a register 640 at physical location R4 as ST(0)--the top of the stack.

Please replace paragraph [0170] with the following:

[0170] As shown in step 750, the packed data instruction is executed without generating any numeric exceptions. Thus, step 750 is similar to step ~~434~~ 440 of FIG. 4B, except the top of stack indication is not altered. As previously described, alternative embodiments which are not completely operating system invisible could be implemented such that either additional event handlers are incorporated into the operating system or existing event handlers are altered to service the errors. If any memory events are generated as a result of attempting to execute the packed data instruction, execution is interrupted and the event is serviced. Of course, an embodiment which did not utilize the EMMS instruction would not require steps 740, 742 and 744.

Please replace paragraph [0171] with the following:

[0171] Thus, a method and apparatus for executing packed data instructions that is compatible with existing operating systems (such as MS-DOS® Windows brand operating environments available from Microsoft® Corporation of Redmond, Wash.) and that promotes good programming techniques is described. Since the packed data state is aliased on the floating point state, the packed data state will be preserved and restored by existing operating systems as if it was the floating point state. Furthermore, since events generated by the execution of the packed data instructions are serviceable by existing operating system event handlers, these event handlers need not be modified and new event handlers need not be added. As a result, the processor is backwards compatible and upgrading does require the cost and time required to develop or modify an operating system.

Please replace paragraph [0195] with the following:

[0195] The rename unit 1004 and the retirement unit 1006 are used to implement register renaming. The technique of register renaming is well known and is performed to avoid storage conflicts resulting from different instructions attempting to use a limited number of storage locations, such as registers. A storage conflict is said to have occurred when such instructions interfere with one another even though the conflicting instructions are otherwise independent. Storage conflicts can be removed by providing additional registers (referred to herein as buffer registers) that are used to reestablish the correspondence between registers and values. To implement register renaming, the processor typically allocates a different one of the buffer registers for every new value produced: that is, for every instruction that writes a register. An instruction identifying the original register--for the purpose of reading its value--obtains instead the value in the allocated buffer register. Thus, the hardware renames the original register identifying the instructions to identify the buffer register and the correct value. The same register identifier in several different instructions may access different hardware registers, depending on the locations of register references with respect to register assignments. For a further description of register renaming, see Johnson, Mike Superscalar Micro Processor Design, 1991 by PTR Prentice-Hall, Inc., New Jersey; "Flag Renaming and Flag Mask Within Register Alias Table," U.S. Ser. No. 08/204,521, now Patent No. 6,047,369, by Colwell, et al.; "Integer and Floating Point Register Alias Table Within Processor Device," U.S. Ser. No. 08/129,678, now Patent No. 5,613,132, by Clift, et al.; and "Partial Width Stalls Within Register Alias Table," U.S. Ser. No. 08/174,841, now Patent No. 5,446,912, by Colwell, et al. When an instruction has successfully completed

execution (without causing any events that are not held pending), the instructions allocated buffer registers are "retired"--the values are transferred from the buffer registers to the original registers identified in the instruction. Alternative embodiments could implement any number of techniques for removing storage conflicts, such as interlocks, partial renaming, etc.

Please replace paragraph [0229] with the following:

[0229] At step 1106, one or more event signal micro ops is inserted indicating the invalid opcode exception should be generated. Event signal micro ops are used to avoid servicing errors until the retirement stage(s) of the pipeline. If an instruction is an event signal micro op, it flows through the decode stage(s), register renaming stage(s), and the execution stage(s). However, when the event signal micro op is received in the retirement stage(s), the state of the buffer registers is not committed and the appropriate event is generated. Event signal micro ops are inserted prior to or in place of the instruction which is causing the event. The use of micro ops is further described with reference to ~~"Method and Apparatus for Signaling an Occurrence of an Event in a Processor," U.S.~~ "Microprocessor With Novel Instruction for Signaling Event Occurrence and for Providing Event Handling Information in Response Thereto," Ser. No. 08/203,790, U.S. Pat. No. 5,625,788, by Darrell D. Boggs, et al. From step 1106, flow passes to step 1108.

Please replace paragraph [0264] with the following:

[0264] As shown in step 1306, the bits [31:0] from those aliased buffer or integer registers are retrieved and flow passes to step 1308. This step is necessary in that the data is stored starting at bit zero. As previously described, in one embodiment this step is performed by the data alignment unit 1090 from FIG. 10. In this embodiment, the data is

transferred from the retirement unit 1006, through the issue unit 1008, and to the execution unit 1010. If the data is accessed from the buffer registers 1020, the data is received by the execution unit 1010 in the format shown in FIG. 12C and the data alignment unit(s) is enabled to extract bits [31:0]. However, if the data is accessed from the integer registers 1024 in an embodiment in which the integer registers 1024 are 32-bit registers, the data is received by the execution unit 1010 in the 32-bit format. In either case, the 32-bits of data may be treated as any of the 64-bits of ~~the~~ a packed data item. For example, a first move instruction could be implemented to move 32 bits from an integer register to the upper bits a packed data item, while a second move instruction could be implemented to move 32 bits from an integer register to the lower 32 bits of a packed data item.

Please replace paragraph [0277] with the following:

[0277] FIG. 15B shows an execution stream, including packed data and floating point instructions, to illustrate the interval of time during which separate physical register files that are aliased may be updated. FIG. ~~15A~~ 15B is similar to FIG. ~~15B~~15A, except a packed data instruction 1530 is followed by a set of floating point instructions 1540. FIG. 15B shows the packed data instruction 1530 is executed at time T1, while the execution of the set of floating point instructions 1540 is started at time T2. Execution of the packed data instruction 1530 causes the processor to write a value to a packed data register. An interval 1550 marks the time between time T1 and time T2 during which this value must be aliased. All of the alternative embodiments described with reference to FIG. 15A (with reference to a floating point instruction followed by packed data instructions) may also be

implemented with reference to FIG. 15B (with reference to a packed data instructions followed by floating point instructions).